AC

Automated software or data updating in distributed computing system

Publication number: GB2348721
Publication date: 2000-10-11
Inventor: TATHAM PAUL (GB); POVAH STEVE (GB)
Applicant: IDEAGEN SOFTWARE LIMITED (GB)
Classification:
- international: ***G06F9/44; G06F9/445; H04L29/06;*** *H04L29/08;* **G06F9/44; G06F9/445; H04L29/06;** H04L29/08; (IPC1-7): G06F9/445; G06F17/30
- European: G06F9/44G4C; G06F9/445N; H04L29/06; H04L29/08N33
Application number: GB20000017328 20000715
Priority number(s): GB20000017328 20000715

Abstract of **GB2348721**

A method and apparatus for automatically updating data or upgrading and maintaining a plurality of client programs in the same version as data or a program stored at a server is disclosed. Each client computer 302, 303 maintains a list of objects comprising a program, which it is currently operating. The server 300 contains a current client program list listing all objects for a current version of client program which is in operation throughout the network. Each client computer refers back to the server, either on boot up, or through a timed periodic check, to make sure the client contains the same list of objects as the server, and to make sure that the client maintains individual objects corresponding to that list in its own memory. By treating programs and data as objects and sending them directly over a network on top of a transfer protocol, efficient and effective real time maintenance of program versions on a plurality of client computers within a network may be achieved.

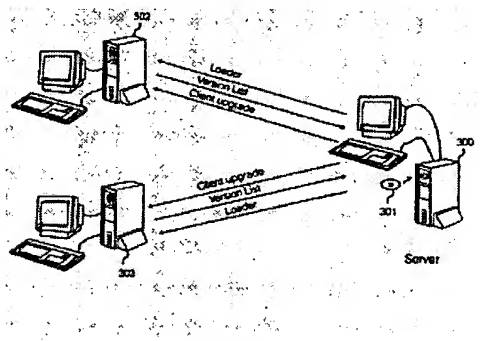

Fig. 3

AUTOMATED SOFTWARE UPGRADE IN DISTRIBUTED COMPUTING SYSTEM

Field of the Invention

The present invention relates to maintenance of logical resources on a plurality of computing entities.

Background to the Invention

As shown in Fig. 1 herein, a typical prior art computer network, for example ranging from a wide area network (WAN) to a local area network (LAN), generally comprises one or a plurality of server computers 100 and a plurality of client computers 101 – 103, connected by a network link 104, operating a standard protocol for example TCP/IP. Client computers may typically comprise personal computers or work stations which operate on a network to execute various application programs. A server computer typically has higher data storage capacity and may have higher data processing capacity, and serves a plurality of client computers by providing programs, files and/or data, and may control hardware devices controlled by clients. Physical links in the local area network can comprise cabling between computer entities, infra-red links, internet connections, or virtual private networks (VPNs).

In the case of corporate networks comprising a plurality of computer entities, upwards of two hundred or so individual computer entities can be connected on a LAN or WAN. One or more servers 100 can include data servers, where the server contains a database making data available to all or a selected number of the client computers, or applications servers, making available an application to all or a selection of the client computers.

Applications programs designed for use in networked computer environments generally comprise a server program resident on a server, and a plurality of client applications, each resident on a client computer. An application

program typically comprises a number of components each of which exist as a separately addressable file, and where each component is identified by a version number. An application program may typically undergo numerous revisions and updates throughout the course of its operational existence on a client computer. In a network environment, an application program on a client computer can be kept current and up to date by replacing one or more of such components, or by adding or deleting components. To upgrade an applications program across a corporate network conventionally involves a human network administrator installing software on the server, typically by reading a software upgrade from a CD-ROM, and also performing a similar operation to install a client software upgrade on each of the plurality of individual client computers. In a corporate network having a large number of computers, say four or five hundred computers, this is a lengthy process as it involves the human administrator visiting each client computer and installing software from a software carrier, eg CD-ROM, at each client computer. In a wide area network, computers may be geographically separated, sometimes in different continents. Implementing an application software upgrade may take a computer administrator hours or days to implement, and implementing a software upgrade across a corporate network may involve downtime on each individual client computer and the server, of the order of upwards of 15 minutes per computer.

The prior art SMS system of International Business Machines provides a system for automatically upgrading software from a server, over a network connection to a plurality of clients. The SMS system comprises a separate application loaded into the server which is activated by a human administrator to download upgrades or other individual applications programs to each of a plurality of client computers from a server. However, this system requires manual intervention by an administrator to activate each upgrade event. Active human management of upgrades from the server is required. The SMS system provides a further layer of network management program to an operating system and a plurality of user applications programs.

In US 6,006,034, there is disclosed an automated method of maintaining application programs on a client computer in a client server network environment involving dynamically upgrading components in the application program running on a client by rendering control to the individual client computer rather than to a central server. A server receives a request for a file from a client, and in response to a request sends a software update by file transfer. However, US 6,006,034 treats individual upgrades as files and requires conversion to a file format of an application program at the client terminal and at the server.

Specific implementations of the present invention aim to provide a method for propagating system upgrades throughout a plurality of client computers from a server computer, quickly.

Summary of the Invention

In a specific implementation according to the present invention, a client-server structured logical entity is provided with a means for self-upgrade between a plurality of computing entities, so that a same version of the logical entity is maintained on each of the plurality of computing entities. The logical entity is preferably an applications program.

In one implementation, a server application maintains a version list recording a current version of the program. Each of a plurality of client applications resident on a corresponding respective client computer, maintains a local version list corresponding to a client application resident at the client computer. Each client application program is revised by addition and replacement of individual files of the application, so that the version of each client application program is maintained to be the same as the version of the server application. A comparison is made between each client version list and the server version list, and any required replacement of individual files comprising a

client application is made by transfer of code over the network dynamically, and automatically without human intervention.

There is provided within the server application a maintenance module. The maintenance module ideally comprises a code component embedded in the server application, which handles deployment of the client programs to the clients computers. On first installation on a server, a loader module is sent via email to each of the plurality of client computers. The application keeps itself up to date in terms of version, across a plurality of client computers in a computer network using maintenance functions provided. Software upgrades are transferred via a transport protocol, for example TCP/IP directly over the network. Individual executable files comprising components of the applications program are treated as objects, which are transferred between the maintenance module and a plurality of loader modules.

As soon as a client computer logs on to a network, the loader module detects whether the client computer has the most up to date version of software, by communicating with the maintenance module. If any changes to the client application version are required, these are downloaded directly from the server computer via the transport link. Because the version upgrades are downloaded directly as objects via the transport link, there is no need for file conversion, which makes upgrade of software faster than in prior art systems. Once the server and client applications are installed on the server and client computers respectively, and the loaders are invoked by activation of the client program, upgrades are automatic and occur without human intervention.

According to a first aspect of the present invention there is provided a network comprising a plurality of computer entities and at least one communications link connecting said plurality of computer entities;

each computer entity comprising:

at least one processor;

a memory device associated with said processor;

said processor and memory device operating in accordance with an operating system;

wherein a first said computer entity is designated as a server computer entity; and

a second said computer entity is designated as a client computer entity, said client computer entity capable of communicating with said server computer entity;

wherein said server computer entity operates to store a current version of a logical entity, and said client computer entity operates to refer to said server computer entity to obtain said current version of said logical entity, said server computer entity communicating said current version of said logical entity to said client computer entity.

According to a second aspect of the present invention there is provided a method of operating a server computer entity for making available a currently held version of a logical entity over a communications interface comprising said server computer entity, said method comprising the steps of:

maintaining a first list of components comprising said logical entity;

receiving from an external source, a second list of objects representing a second logical entity;

comparing said first list of objects with said second list of objects; and

creating an instruction message containing instructions to modify said second list of objects to be the same as said first list of objects.

According to a third aspect of the present invention there is provided means for maintaining a logical entity upon a network, said network comprising a plurality of computer entities capable of communicating with each other, said means comprising:

a first list of object components comprising said logical entity;

means for comparing said first list of components with a second list of components;

means for creating a message depending on a result of said comparison of said first component list with said second component list, said message comprising a set of instructions for maintaining said second component list to be the same as said first component list.

According to a fourth aspect of the present invention there is provided a method of operating a client computer entity for maintaining a client logical entity stored on said client computer entity, in a current version, said method comprising the steps of:

receiving a first component list describing a list of components comprising a current version of a server host logical entity;

maintaining a second list of components comprising said client logical entity stored on said client computer entity;

comparing said second client component list with said first component list; and

if said first component list differs from said second component list, modifying said second component list to correspond with said first component list.

According to a fifth aspect of the present invention there is provided a method of operating a server computer entity for making available a currently held version of a logical entity over a communications interface comprising said server computer entity, said method comprising the steps of:

maintaining a list of components comprising a current version of said logical entity;

selecting a plurality of client computer entities to which said current version logical entity is to be sent;

sending to each said selected client computer entity, a loader means, said loader means being capable of communicating with said server computer entity for maintaining said logical entity on a said client computer, in a same version as on said server computer entity.

Other aspects of the present invention are as recited in the claims herein.

Brief Description of the Drawings

For a better understanding of the invention and to show how the same may be carried into effect, there will now be described by way of example only, specific embodiments, methods and processes according to the present invention with reference to the accompanying drawings in which:

Fig. 1 illustrates schematically a prior art network of a plurality of computing entities communicating by means of a network communications link;

Fig. 2 illustrates schematically a client computer and server computer comprising part of a network of a plurality of computer entities, including a program maintenance system according to a specific implementation of the present invention;

Fig. 3 illustrates schematically a communication between a server computer and a plurality of client computers for upgrade of a client program from the centralized server accessible by all client computers;

Fig. 4 illustrates schematically an installation routine for initial installation of an application program on a server computer and a plurality of client computers;

Fig. 5 illustrates schematically a logical layout of a server program and a plurality of loader programs resident on a plurality of client computer entities according to the specific implementation of the present invention;

Fig. 6 illustrates schematically an installation mode of a client loader program for loading a client host program from a server on first installation;

Fig. 7 illustrates schematically an architecture of the loader program of Fig. 6;

Fig. 8 illustrates schematically an architecture of a maintenance program loaded onto the server computer;

Fig. 9 illustrates schematically a client version list maintained at a client computer for ensuring a current version of a client host program is maintained on the client computer;

Fig. 10 illustrates schematically a server version list maintained at a server computer for making sure that a current client host program is maintained in a current version at each of a plurality of computer entities in a network;

Fig. 11 illustrates schematically an overall operation of a first mode of operation of the server for maintaining a plurality of client host programs in a current version as loaded at the server computer entity;

Fig. 12 illustrates schematically a server version message sent over a network directly as an object over a transport protocol for messaging an individual client computer entity with a most up to date version of client host program;

Fig. 13 illustrates schematically a client version list scan operation for scanning a received client version list from a client computer entity, carried out at the server computer;

Fig. 14 illustrates schematically a scan server version list for comparing a server version list of a client computer program with a client version list of a client computer program carried out at the server computer entity;

Fig. 15 illustrates schematically a second mode of operation of a client computer for maintaining a client host program at the client computer with a current client host program maintained at a server and identified by receipt of a server version message from the server; and

Fig. 16 illustrates a third mode of operation of a client computer for obtaining object code or data from a server computer, in order to maintain a client host program in a same version as the client host program stored at the server.

Detailed Description of the Best Mode for Carrying Out the Invention

There will now be described by way of example the best mode contemplated by the inventors for carrying out the invention. In the following description numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent however, to one skilled in the art, that the present invention may be practiced without limitation to these specific details. In other instances, well known methods and structures have not been described in detail so as not to unnecessarily obscure the present invention.

According to a specific implementation of the present invention, in a network of computer entities comprising at least one server computer and a plurality of client computers communicating with each other via a network, on the server, is provided a server logical entity serving a plurality of client logical entities, each resident on a corresponding respective client. Within the server logical entity is provided a maintenance means for maintaining each client entity in a same version as the server entity. In each client is provided a loader means for loading a logical entity version from the sever.

In this specification, by "logical entity" it is meant any unit of data or program code which is capable of direct transport across a transport layer of a network connection, and the term includes but is not limited to: individual applications programs; data stored in a database; computer programming languages; architectural features of data, programs or computer programming languages.

For ease of explanation hereafter, there will be described an implementation in which the logical entity is an applications program. However, it will be understood by those skilled in the art that the methods, processes and apparatus described herein apply in principle equally to any other logical entity.

The maintenance means comprises a server module embedded within the server application, and a plurality of client modules, each embedded within a client application.

Referring to Fig. 2 herein, there is shown schematically a single server computer 200 and a single client computer 201. The server computer comprises a communications port 202 for communicating over a local area network; a processor 203, with associated memory 204; non-volatile data storage 205, for example a hard disk or RAID array; an operating system 207 for example the known Windows NT®, Windows 2000®, LINUX® or UNIX® operating systems; and one or a plurality of server applications 208 – 210, each containing a server module for maintaining the application according to a specific implementation of the present invention. Each client computer comprises a communications port 211 for communicating with the network; a processor 212 with associated memory 213; a non-volatile data storage device, eg a hard disk drive 214; optionally, a user interface 215; an operating system 216, for example Windows NT®, Windows 2000®, LINUX® or UNIX ®; and one or a plurality of client applications 217 – 219, at least one client application containing a client module in accordance with a specific implementation of the present invention as described hereafter.

Referring to Figs 3 to 5 herein, there will now be described in overview an overall structure and operation of a maintenance method and apparatus of the best mode specific implementation.

Referring to Fig. 3 and 4 herein, there is illustrated schematically an overall operation of the maintenance means according to the best mode implementation. Server computer 300 is initially installed with a maintenance module according to the specific implementation of the present invention, by loading from a data carrier 301, for example a CD-ROM or the like, the maintenance module into the server. The maintenance module is carried within the application, which acts as

a host carrier program, and is integral to the carrier program. The server maintenance module serves the purpose of maintaining and upgrading the particular carrier program to which it has been written into. Initially, a human operator installs a server application 208 on the server computer in step 400. This involves installation in a conventional manner, for example the user may install the server application from one or a plurality of CD-ROM data carriers using a CD-ROM port on the server computer 200. Installation is carried out under control of an installation module within the application, as is known in the art. The server maintenance module, having been installed into the server in conventional manner from data carrier 301, in step 400 immediately activates to send a loader module to each of the client computers in step 401. The loader module comprises an application program for installation on each client computer. The loader application may be sent by email to each of the client computers 302, 303. A human operator writes emails to each client and attaches the loader application as a file. A human network manager at the server, may, upon loading the host client carrier program containing the maintenance module, select which of a plurality of client computers the carrier program is to be loaded upon in step 401. For each selected client computer, in step 402 a loader application is sent via email over the network. In step 403, the loader application is activated at each client computer. Activation of the loader application typically will occur on rebooting the computer. For example, when a client computer first receives the loader application via email, a user at the client computer may open the email and load the loader module onto the client computer. The application is self-maintaining, by virtue of the server module and loader application communicating with each other such that the client module refers to the server module to check that the version of client application is the same as the current version of server application. If the version of client application deviates from the version of server application, the server sends a new client application version to each client. Once activated, the loader application at each client continues to maintain the host carrier program in a current version at that particular client computer on which it is resident, by continuously and periodically referring to the

server over the network to make sure that the set of files comprising the carrier program which are in operation at the client computer match those which are specified by the reference program at the server. Validation of a client's logical entities occur asynchronously whenever a loader application of that client is run.

Referring to Fig. 5 herein, there is illustrated schematically a logical representation of the host carrier program once installed on the plurality of networked computers. The host server program 500 acting as a carrier to the maintenance module 501 is resident on the server computer. Each of the client computers has resident a client host program 502, which has been loaded by client loader application 503. Client loader application 503 and maintenance program 501 ensure that every time the server host program is upgraded, each of the client host programs 502 are maintained with the same version as the server host program. The maintenance module and loader application are best implemented as code embedded in the host application program. The maintenance module and loader applications may be provided to applications developers for inclusion into new applications written. The maintenance and loader applications may be provided on a data carrier such as a CD ROM, and may be implemented in a suitable programming as known in the art, for example Delphi. The maintenance module 501 sends the client program 502 to each deployed loader application 503.

Referring to Fig. 6 herein, there is illustrated a first method of operation of a client loader module 503. In step 600, the client loader searches for a server containing the server module. In step 601, if the server location is known to the client loader module, then in step 603 the client loader connects directly to the server module. However, if the server location is not known to the client loader, in step 602 the client loader asks a user to enter a server location. This may be by prompting a user at a client terminal to enter a location of a server, or to select a server from an icon display of computers at the client terminal. In step 604,

having located the address of the server, the client loader sends its current version list to the server.

The method as illustrated in Fig. 6 activates every time the client loader is invoked.

Referring to Fig. 7 herein, there is illustrated schematically components of loader module 503. Loader application 503 comprises a communications interface 701 for communicating with the server maintenance module 501 over a network; a client management module 702 for managing the client host program 502; a file system interface 703 for interfacing with a local file system of the client computer; and a client version list 704 comprising a list of program components comprising the client host program 502 stored on the client computer. All program components are treated as named objects in the client version list. Each name in the client version list 704 relates to a corresponding respective component of code comprising client host program 502.

Referring to Fig. 8 herein, there is illustrated schematically components of maintenance module 501 resident in server host program 500 on the server computer. Maintenance module 501 comprises communications interface 800 for interfacing with the communications port of the server computer; a server management module 801 operating to manage a plurality of client host programs resident on a plurality of client computers in the network; a file system interface 802 capable of interfacing with a local file system on a server operating system of the server computer; and a server version list 803 comprising a list of code components for a latest version of client program. Each entry in the server version list relates to a set of code stored on the server, and which can be distributed over the network to any of the client computers to which the loader application 503 has been installed. Within the maintenance program, each component of code of the latest version of the client host program is treated as

an object. Each object is referred to in the server version list 803 for example by a file name.

Referring to Fig. 9 herein, there is illustrated an example of a client version list 704 resident on a client computer. The client version list comprises a set of file names, each referring to a code component resident on the client computer. For example, in Fig. 9 there is illustrated a plurality of dynamic link library files 900, each having a name and a version value. Similarly there is illustrated a plurality of help files 903 also each having a file name 904 and a version value 905. Since any application program generally comprises a plurality of individual files, each file comprising a discrete quantity of code and referenced by a file name, a particular version of an application comprises a set of such files. Therefore, installing an upgrade of a particular client host program comprises storing a different set of component code files on the client computer.

Referring to Fig. 10 herein, at the server computer, server version list 803 comprises a list of objects comprising a latest version of a client program implemented on the network. The server version list comprises a list of individual file names, each referring to a code component. The complete list of file names, which are stored as objects, comprises a complete listing of a client program in operation on an individual client computer. All individual components of code referred to in the server version list 803 comprise the current version of the client host program.

When a client program is upgraded, upgrades can be entered at the server by a human network administrator in conventional manner, for example by downloading them from a data storage carrier, eg a CD-ROM. With the latest version of the client program, there is provided a list of individual code components making up the client program. These are stored as the server version list 803 in the file system of the operating system run by the server computer. Since the individual code components in the latest version of the client

program at the server may not correspond to earlier versions of the client program which may be operational at individual ones of the plurality of client computers, the server management module 801 has various modes of operation for maintaining the individual client host programs at each of the individual client computers to be the same version as present on the server. For example, the server version list illustrated in Fig. 10 shows different client host program components to the client version list shown in Fig. 9. Under such circumstances management module 801 would operate to update the client computer with the version of the client host program stored on the server computer.

Referring to Fig. 11 herein, there is illustrated schematically in overview a first mode of operation of the server on receiving a client version list from a client computer over a network connection in step 1100. In step 1101, the server scans the client version list by inspecting individually in turn each object name in the client version list. Typically these are file names, for example help files, executable files, DLL files or the like. In step 1102, the server scans the server version list containing the current version of objects present in the current version of the client's host program. In step 1103, any code components or data components which the server has determined are missing from the individual client computer are sent over the network as objects, using the transport protocol, for example TCP/IP, and are received by the client computer. In step 1104, the server sends a server version message to the client computer. The server version message comprises a plurality of object instances, which together make up the host client computer. The client computer stores this list as its own client version list, and modifies its own stored objects so that they correspond with the named types on the client version list.

Once the client computer has stored a client version list which is the same as the server version list, and the client computer has stored each individual object type referred to in the client version list, then the client host program stored on the client computer is the same as the version stored on the server.

Referring to Fig. 12 herein, there is illustrated schematically a server version message sent from the server to a client computer. The server version message comprises a list of types of object, in this case code files, identifying each object by name 1102, and for each object providing an instruction 1103 to the client computer on how to treat that object.

Instructions to the client computer include a **REMOVE** instruction, instructing the client computer to remove an object of that type from its client version list; an **ADD** instruction, instructing the client computer to add an object of that type to its client version list; a **RETAIN** instruction, instructing the client computer to retain an object of that name within its client version list.

In the example of Fig. 12, having compared a received client version list of Fig. 9 with a stored server version list of Fig. 10, the server generates a server reply message as shown in Fig. 12 having the instructions to remove, add and retain various files in order that, upon receipt of the message, the client computer will amend its own client version list to be identical to that of the server version list.

Referring to Fig. 13, there is illustrated in more detail the process for scanning a received client version list operated at the server according to a first server mode of operation. Each of the objects in the client list received by the server are inspected in turn by the server in step 1301. In step 1302, the object entry in the client version list is compared with the current server version list. If the object in the received client version list corresponds with that in the current server version list, then in step 1303 the server adds a **RETAIN** message to a server version message in respect of that object. However, if in step 1302 the object entry in the client version list does not correspond with an identical object in the current server version list, then in step 1304 the server adds a **REMOVE** message for the object to the server version message to be sent to the client

computer. In step 1305, the server inspects further objects in the client version list. If not all objects in the client version list have been processed, then the server repeats steps 1301 – 1305. However, if in step 1305 all objects in the client version list have been processed, then in step 1306 it is checked whether all objects in the server version list have been processed. If all objects in the server version list have not been processed, then in step 1400 the server scans the server version list. However, if in step 1306 all objects in the server version list have been processed then in step 1307 the server version message which has been compiled by the server is sent to the client computer.

Referring to Fig. 14 herein, is shown an operation of the server computer for scanning the server version list for comparing the server version list with a received client version list. In step 1401, each server list object entry is processed by the server for the purpose of comparing objects in the current server version with those in the received client version list. In step 1402, each object in the client version list is inspected to see if the current object in the server version list exists in the client version list. If the currently inspected object in the server version list does correspond with an identical object in the received client version list, the in step 1403 the server adds a **RETAIN** message to the server version message for the particular object being inspected. However, if in step 1402 the currently inspected object in the server version list does not correspond to an identical object in the received client version list, then that object is sent to the client computer in step 1404 over the communications network, directly using a transport protocol, for example TCP/IP. In step 1405, the server adds an **ADD** message to the server version message list for that object. The server version message is stored locally at the server until it is complete and ready to be sent to the client computer. In step 1406, it is checked whether all objects in the server version list have been processed. If not, then steps 1401 – 1405 are repeated. However, if in step 1406 all objects in the server version list have been inspected, then in step 1407 it is checked whether all objects in the received client version list have been processed. If not all objects in the received client version list have

been processed, then in process 1300 the client version list is scanned as described with reference to Fig. 13 hereinbefore. Once all objects in the received client version list and all objects in the maintained current server version list have been inspected, then the completed server version message is sent to the client computer in step 1408.

The first mode of operation of the server as shown in Figs. 11 - 14 herein is activated by receipt of a client version list by an individual client computer. The first mode of operation shown in Figs. 11 – 14 operates sequentially or in parallel, for each of the plurality of client computers having a client host program corresponding to the server host program.

Referring to Fig. 15 herein, there is illustrated schematically a second mode of operation, operated by a client computer upon receiving a server version message. The received server version message is scanned in step 1500, each object entry in the server message 1501 is checked to read an object name, and an instruction corresponding to the object name. In step 1502, if the instruction is **REMOVE**, the in step 1503 an object corresponding to that object name is removed from the currently stored client version list at the client computer. In step 1504 if the instruction is **ADD**, then in step 1505 an object name corresponding to that in the object entry in the server version message is added to the client version list. If, in step 1506, the instruction is **RETAIN**, then in step 1507 an object with that name is retained in the client version list, and also retained in memory. Once all objects in the server version list received in the server version message have been inspected in step 1508, then the client version list will have been updated to be the same as the client version list on the server.

In parallel, or earlier than carrying out the third mode of operation of Fig. 15 at the client computer, the client computer may have received individual objects

having object names referred to in the received server version message, and may have previously stored these on its disk.

Referring to Fig. 16, according to a third mode of operation, the client computer maintains its stored objects on disk to be the same as the named objects in the client version list, the client version list being maintained by the first and second modes of operation to be the same as the server version list.

In step 1600, the client computer scans each entry in the client version list. The client version list contains a list of file names and version numbers, together with a stored instruction on the retention, addition or removal of a particular object.

In step 1601, a first client list object entry is inspected. In step 1602, if the object already exists in the client file system in memory, corresponding to the object listed in the client version list, then there is no need to take any further action with respect to that object. However, if in step 1602 an object exists in the current client version list, which is not actually stored on disk of the client computer, then in step 1603, the client computer sends a request message to the server to send that particular object over the network, so that the client computer can maintain a number of objects in memory which correspond to its currently stored client version list.

Whilst in this specification operation of a maintenance system has been illustrated with reference to a host server application and a host client application program, it will be understood by those in the art that since it is objects which are transferred directly over the network via an underlying transport protocol, such as TCP/IP, the scope of the invention is not limited to maintenance of application programs, but can be applied to any data which can be treated in object oriented manner. For example upgrades of data in a database may be treated as different

versions of data and may be sent as object across a computer network directly via an underlying transport protocol.

Claims:

1. A network comprising a plurality of computer entities and at least one communications link connecting said plurality of computer entities;

each computer entity comprising:

at least one processor;

a memory device associated with said processor;

said processor and memory device operating in accordance with an operating system;

wherein a first said computer entity is designated as a server computer entity; and

a second said computer entity is designated as a client computer entity, said client computer entity capable of communicating with said server computer entity;

wherein said server computer entity operates to store a current version of a logical entity, and said client computer entity operates to refer to said server computer entity to obtain said current version of said logical entity, said server computer entity communicating said current version of said logical entity to said client computer entity.

2. The network as claimed in claim 1, wherein said logical entity comprises a plurality of individual components, and said server computer entity communicates at least one said individual component as an object.

3. The network as claimed in claim 1, wherein said current version of said logical entity is communicated directly within a transport layer protocol operating between said first and second computer entities.

4. The network as claimed in claim 1, wherein said server entity communicates a current version of said logical entity to said client computer entity, by sending a first object list, said first object list describing a plurality of components of said logical entity.

5. The network as claimed in claim 1, wherein said client computer entity refers to said server computer entity by sending a second list of objects to said server computer entity, said second list of objects describing a current version of a client logical entity residing on said client computer entity.

6. The network as claimed in claim 1, wherein said server computer entity maintains a plurality of components of said current version of said logical entity.

7. The network as claimed in claim 1, wherein said client computer entity comprises a loader means, said loader means operating to refer to said server computer entity to obtain said current version of said logical entity.

8. The network as claimed in claim 1, wherein said client computer entity refers to said server computer entity, upon invocation of a loader means resident in said client computer.

9. The network as claimed in claim 1, wherein said server computer entity comprises a maintenance means, said maintenance means operating to:

maintain said current version of said logical entity at said server computer entity; and

communicate with said client computer entity to maintain a client version of said logical entity on said client computer entity.

10. A method of operating a server computer entity for making available a currently held version of a logical entity over a communications interface comprising said server computer entity, said method comprising the steps of:

maintaining a first list of components comprising said logical entity;

receiving from an external source, a second list of objects representing a second logical entity;

comparing said first list of objects with said second list of objects; and

creating an instruction message containing instructions to modify said second list of objects to be the same as said first list of objects.

11. The method as claimed in claim 10, wherein said step of creating an instruction message comprises:

for each object of said second list which is identical to a corresponding respective object of said first list, generating a RETAIN message, for retaining said object in said second object list.

12. The method as claimed in claim 10, wherein said step of creating an instruction message comprises:

for each object of said first list which is not present in said second list, generating an ADD message for adding said object to said second object list.

13. The method as claimed in claim 10, wherein said step of creating an instruction message comprises:

for each object of said second list which does not appear in said first list, generating a REMOVE message to delete said object from said second object list.

14. The method as claimed in claim 10, further comprising the step of:

for each object of said first list which is not present in said second list, sending said object via said communications interface.

15. The method as claimed in claim 14, wherein said object is carried directly within a transport protocol via said communications interface.

16. Means for maintaining a logical entity upon a network, said network comprising a plurality of computer entities capable of communicating with each other, said means comprising:

a first list of object components comprising said logical entity;

means for comparing said first list of components with a second list of components;

means for creating a message depending on a result of said comparison of said first component list with said second component list, said message comprising a set of instructions for maintaining said second component list to be the same as said first component list.

17. The means as claimed in claim 16, further comprising:

means, for loading said logical entity onto a client computer entity, said loader means comprising:

means for maintaining said second list of components on said client computer entity; and

means for requesting transfer of at least one said component between individual ones of said plurality of computer entities.

18. A method of operating a client computer entity for maintaining a client logical entity stored on said client computer entity, in a current version, said method comprising the steps of:

receiving a first component list describing a list of components comprising a current version of a server host logical entity;

maintaining a second list of components comprising said client logical entity stored on said client computer entity;

comparing said second client component list with said first component list; and

if said first component list differs from said second component list, modifying said second component list to correspond with said first component list.

19. The method as claimed in claim 18, wherein maintaining said client logical entity in a current version comprises:

adding at least one logical component to said client computer entity.

20. The method as claimed in claim 18, wherein maintaining said client logical entity in a current version comprises removing at least one logical component from said client computer entity.

21. The method as claimed in claim 18, wherein maintaining said client logical entity in a current version further comprises the steps of:

storing a plurality of components as listed on said second component list at said client computer entity; and

if a component listed in said second component list is not present on said client computer entity, sending a request message, requesting sending of said missing component.

22. The method as claimed in claim 18, wherein said step of modifying said second component list comprises:

adding a component to said second component list.

23. The method as claimed in claim 18, wherein said step of modifying said second component list comprises:

removing a component from said second component list.

24. The method as claimed in claim 18, wherein said method is activated on invocation of a loader means in said client computer entity.

25. A method of operating a server computer entity for making available a currently held version of a logical entity over a communications interface comprising said server computer entity, said method comprising the steps of:

(12) UK Patent Application (19) GB (11) 2 348 721 (13) A

(43) Date of A Publication 11.10.2000

(21) Application No 0017328.6

(22) Date of Filing 15.07.2000

(71) Applicant(s)
IdeaGen Software Limited
(Incorporated in the United Kingdom)
Limetree Business Park, MATLOCK, Derbyshire,
DE4 3EJ, United Kingdom

(72) Inventor(s)
Paul Tatham
Steve Povah

(74) Agent and/or Address for Service
Franks & Co
352 Omega Court, Cemetery Road, SHEFFIELD,
S11 8FT, United Kingdom

(51) INT CL[7]
G06F 9/445 17/30

(52) UK CL (Edition R)
G4A AFL AUDB

(56) Documents Cited
GB 2341462 A GB 2333864 A GB 2325766 A
GB 2203573 A EP 0841615 A2 EP 0284924 A2
WO 99/56207 A1 WO 91/02313 A1 US 6009274 A
US 5835911 A

(58) Field of Search
UK CL (Edition R) G4A AFL AUDB
INT CL[7] G06F 9/44 9/445 15/177 17/30
Online: EPODOC, Internet, JAPIO, WPI

(54) Abstract Title
Automated software or data updating in distributed computing system

(57) A method and apparatus for automatically updating data or upgrading and maintaining a plurality of client programs in the same version as data or a program stored at a server is disclosed. Each client computer 302, 303 maintains a list of objects comprising a program, which it is currently operating. The server 300 contains a current client program list listing all objects for a current version of client program which is in operation throughout the network. Each client computer refers back to the server, either on boot up, or through a timed periodic check, to make sure the client contains the same list of objects as the server, and to make sure that the client maintains individual objects corresponding to that list in its own memory. By treating programs and data as objects and sending them directly over a network on top of a transfer protocol, efficient and effective real time maintenance of program versions on a plurality of client computers within a network may be achieved.

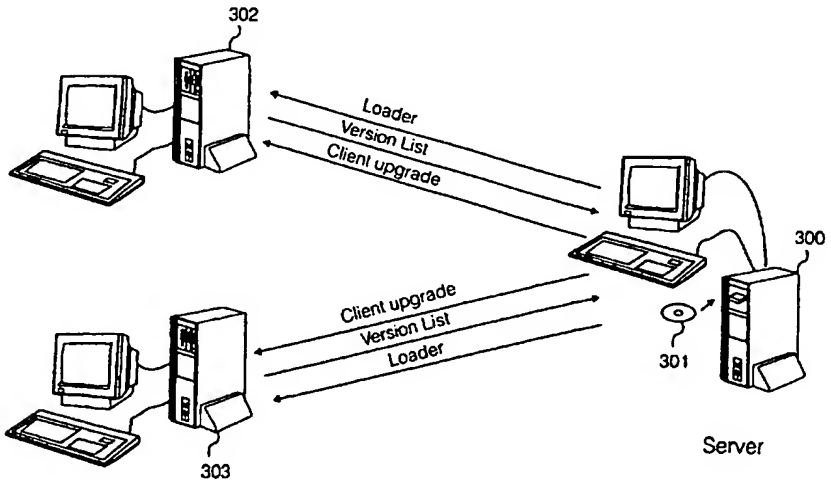

Fig. 3

GB 2 348 721 A

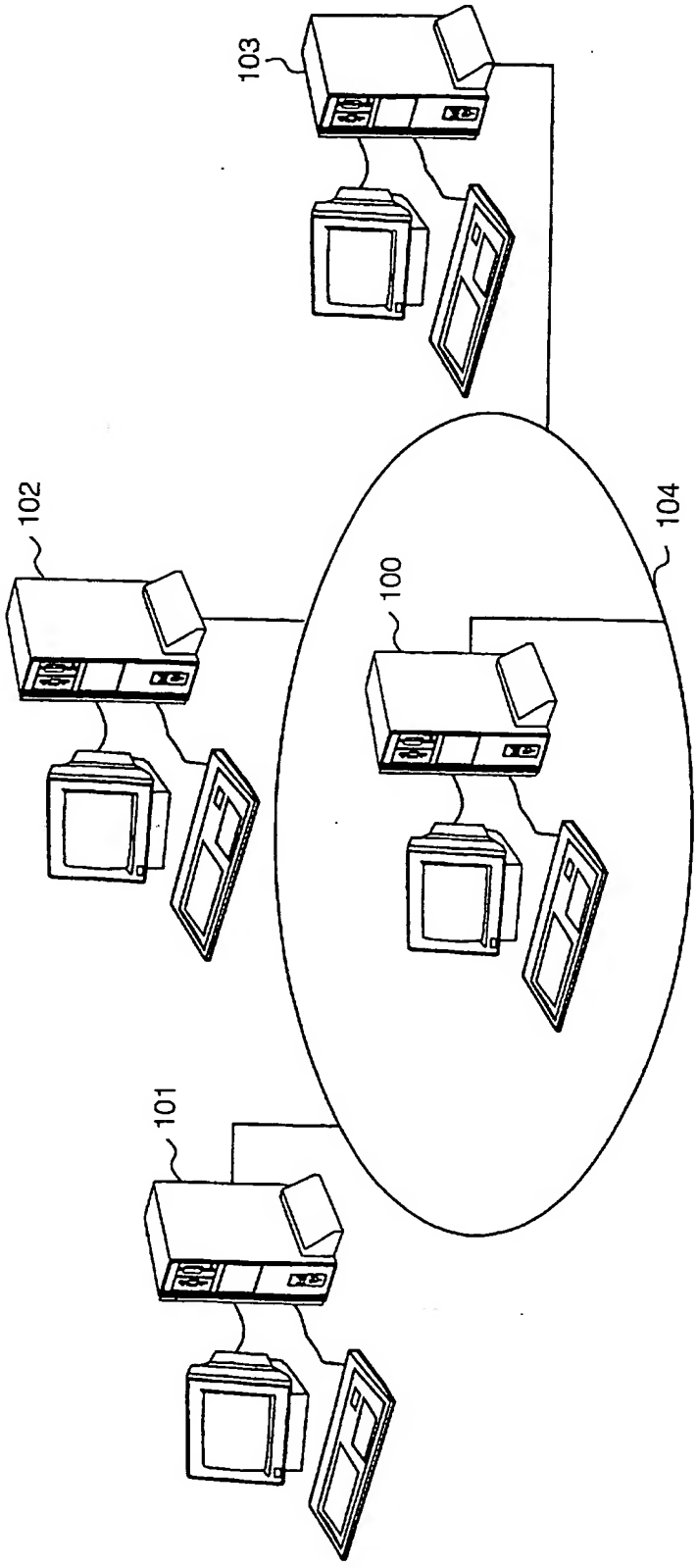

Fig. 1
(Prior Art)

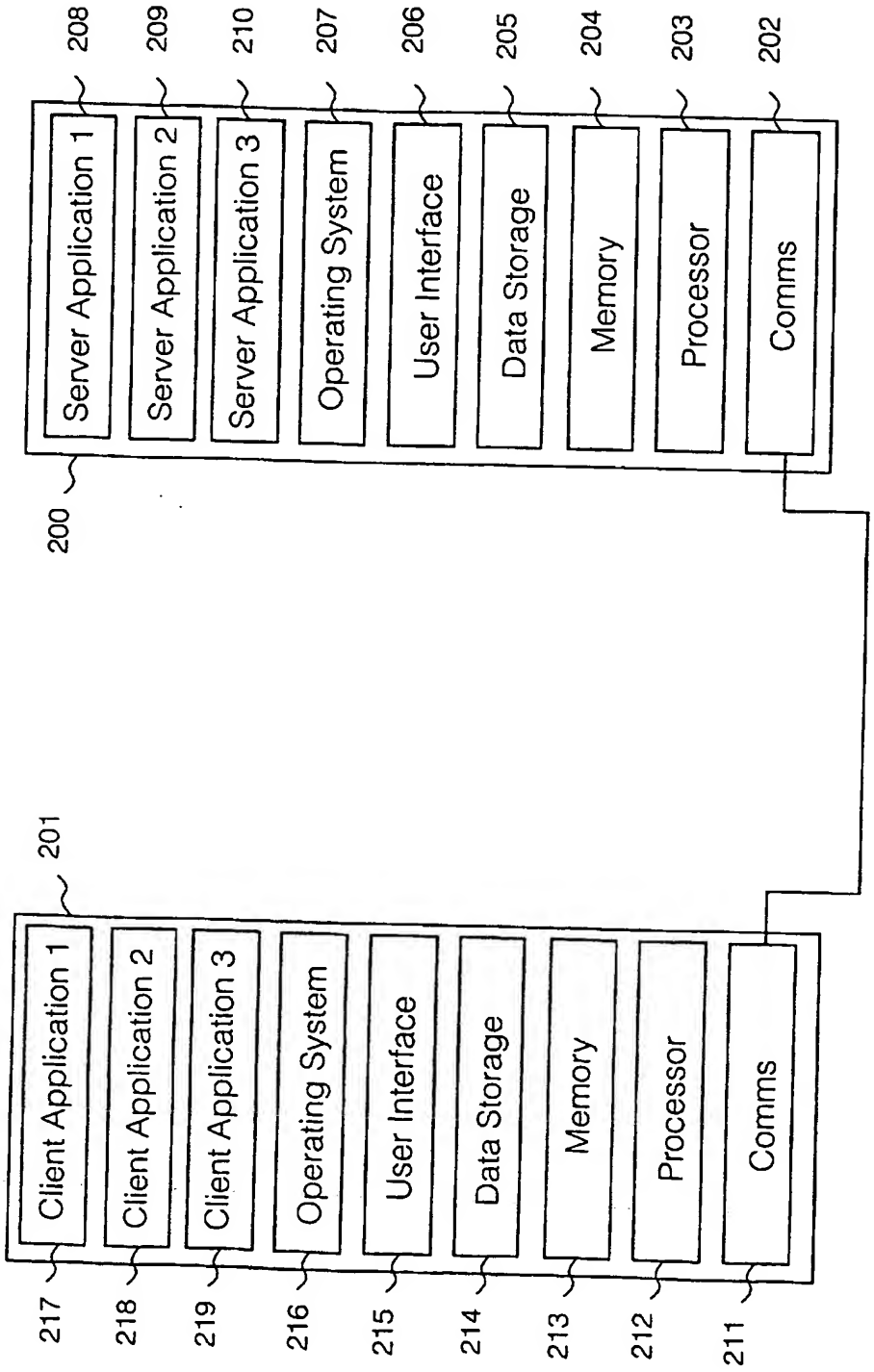
Server Application 1
Server Application 2
Server Application 3
Operating System
User Interface
Data Storage
Memory
Processor
Comms
208
209
210
207
206
205
204
203
202
200
Server
Client Application 1
Client Application 2
Client Application 3
Operating System
User Interface
Data Storage
Memory
Processor
Comms
201
217
218
219
216
215
214
213
212
211
Client

Fig. 2

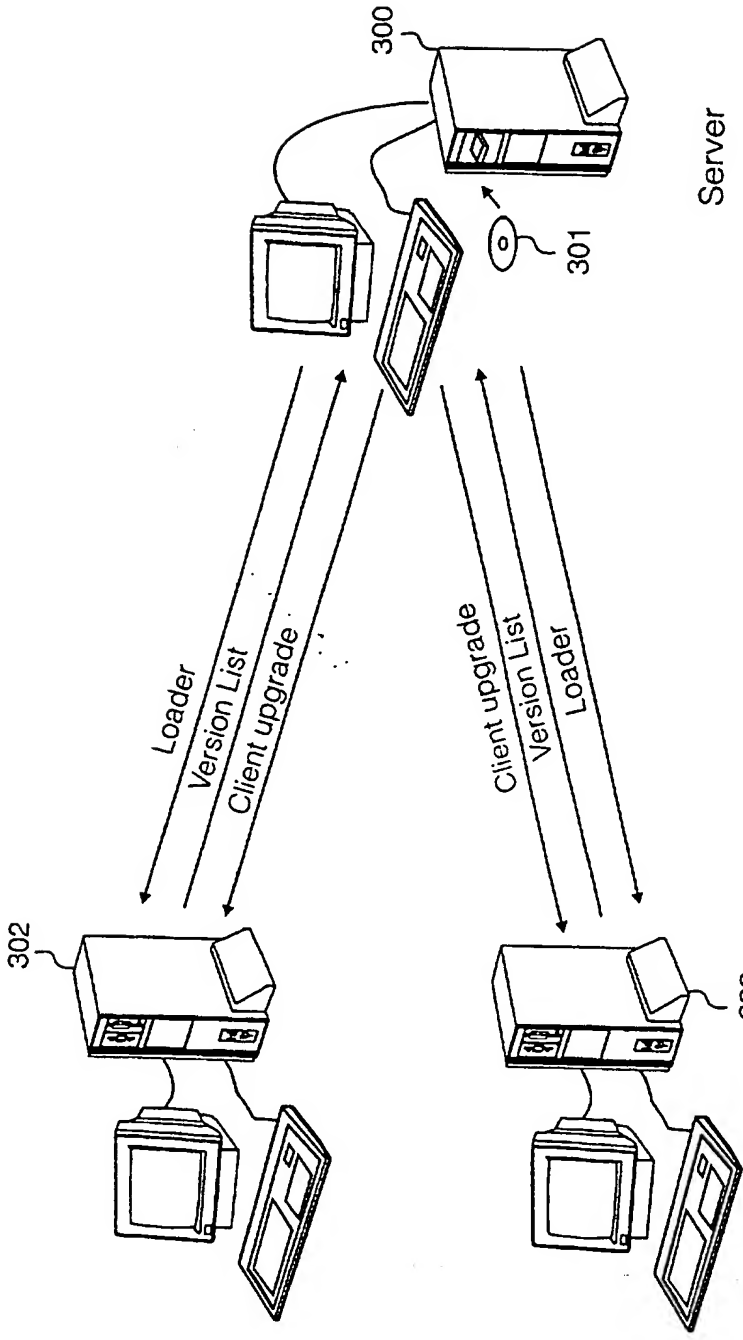
300
Server
301
Loader
Version List
Client upgrade
Client upgrade
Version List
Loader
302
303

Fig. 3

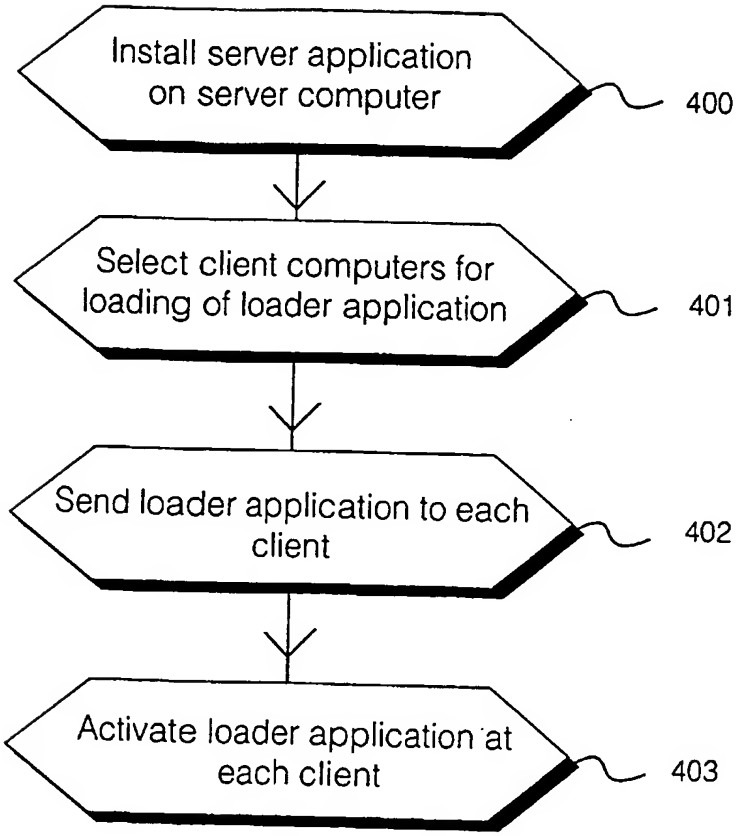
Install server application on server computer
400
Select client computers for loading of loader application
401
Send loader application to each client
402
Activate loader application at each client
403

Fig. 4

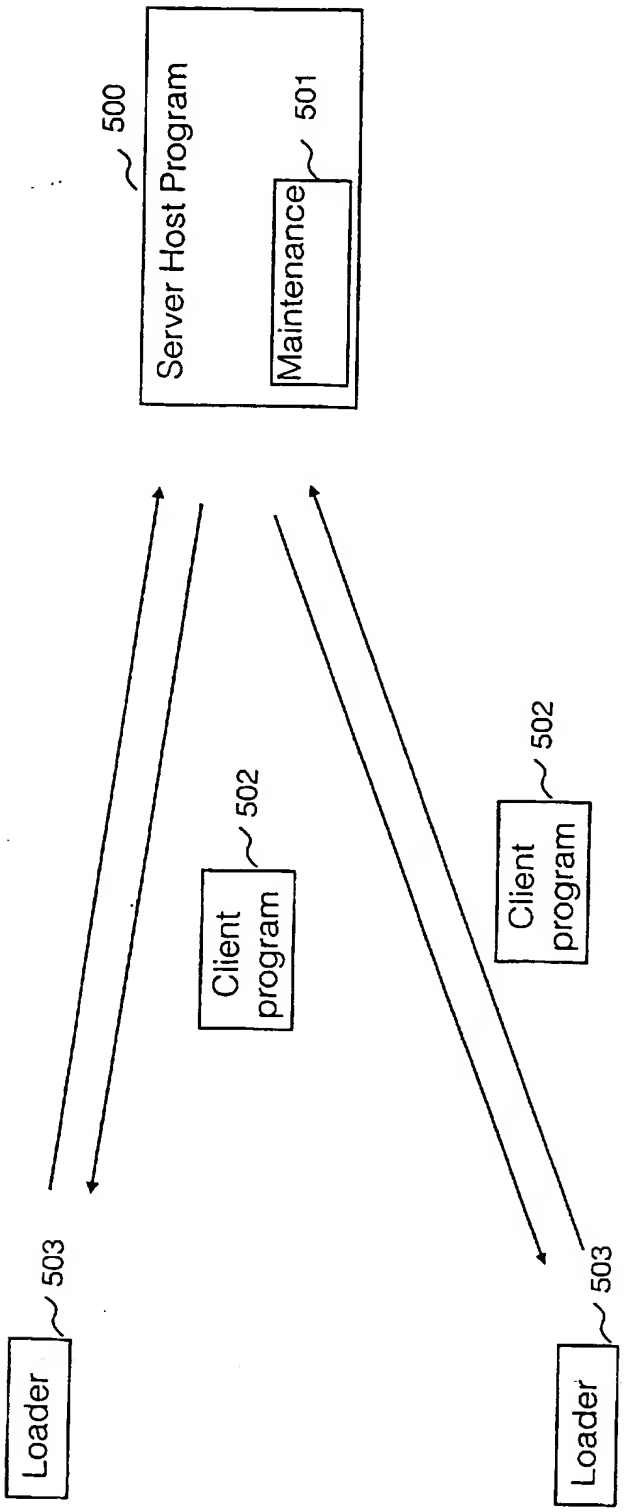
500
Server Host Program
501
Maintenance
502
Client program
502
Client program
503
Loader
503
Loader

Fig. 5

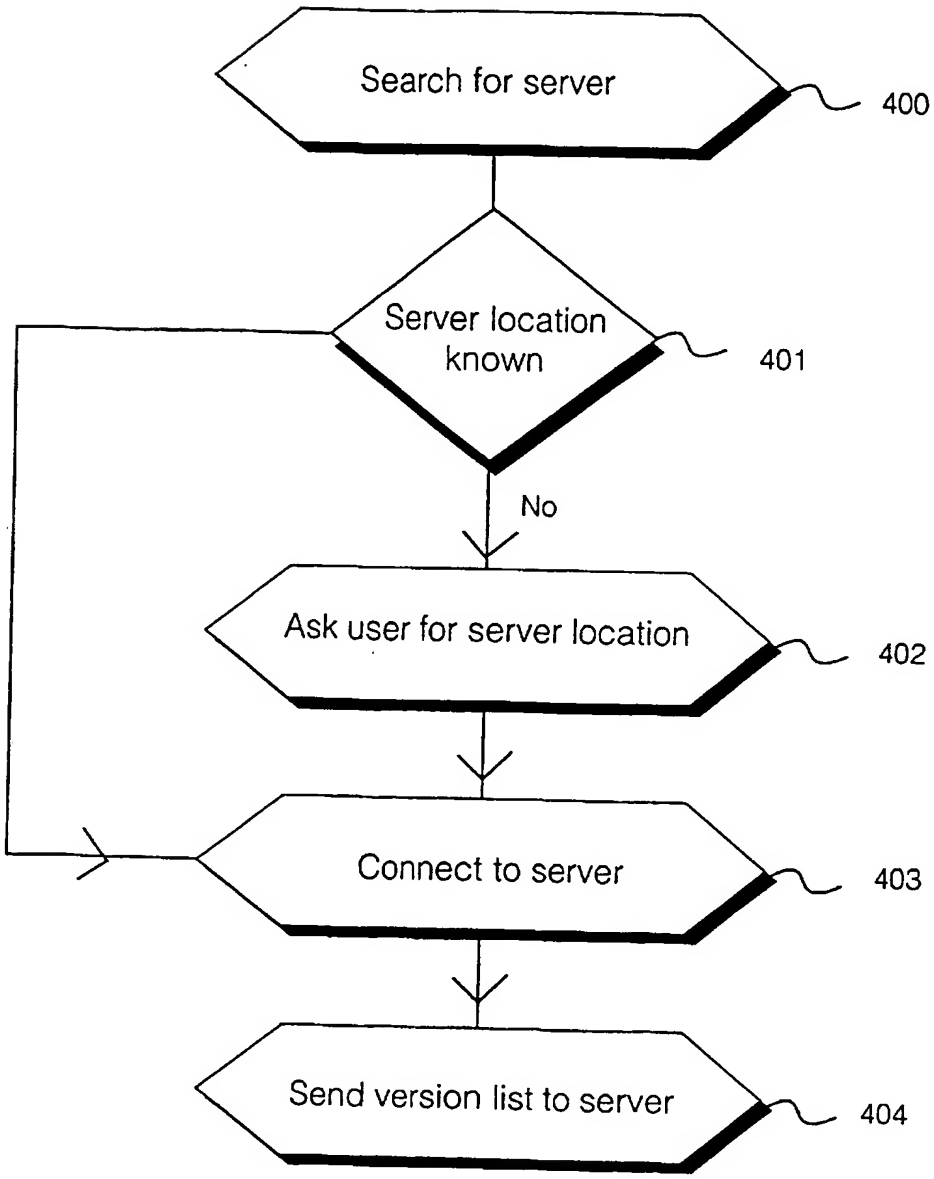
Search for server
400
Server location known
401
No
Ask user for server location
402
Connect to server
403
Send version list to server
404

Fig. 6

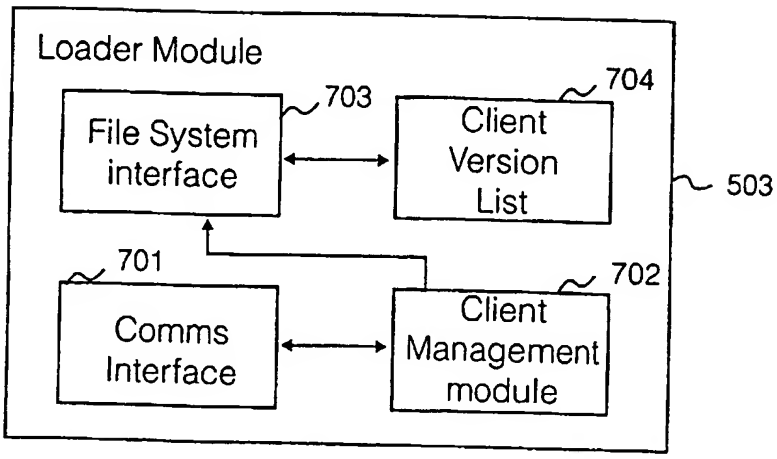
Loader Module
703
704
File System interface
Client Version List
503
701
702
Comms Interface
Client Management module

Fig. 7

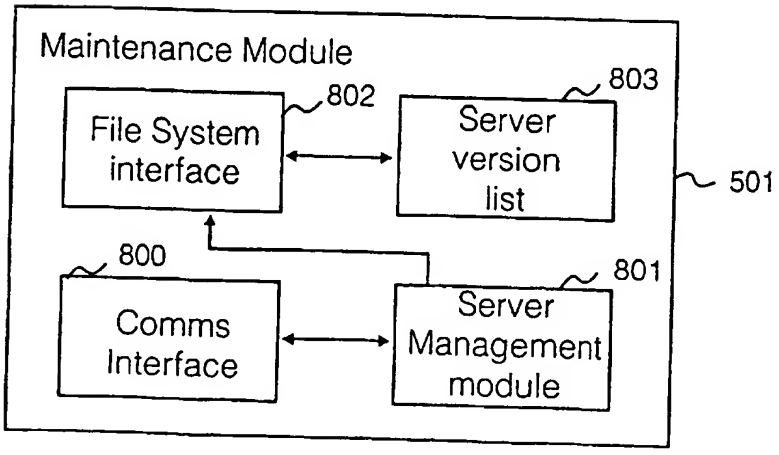
Maintenance Module
802
803
File System interface
Server version list
501
800
801
Comms Interface
Server Management module

Fig. 8

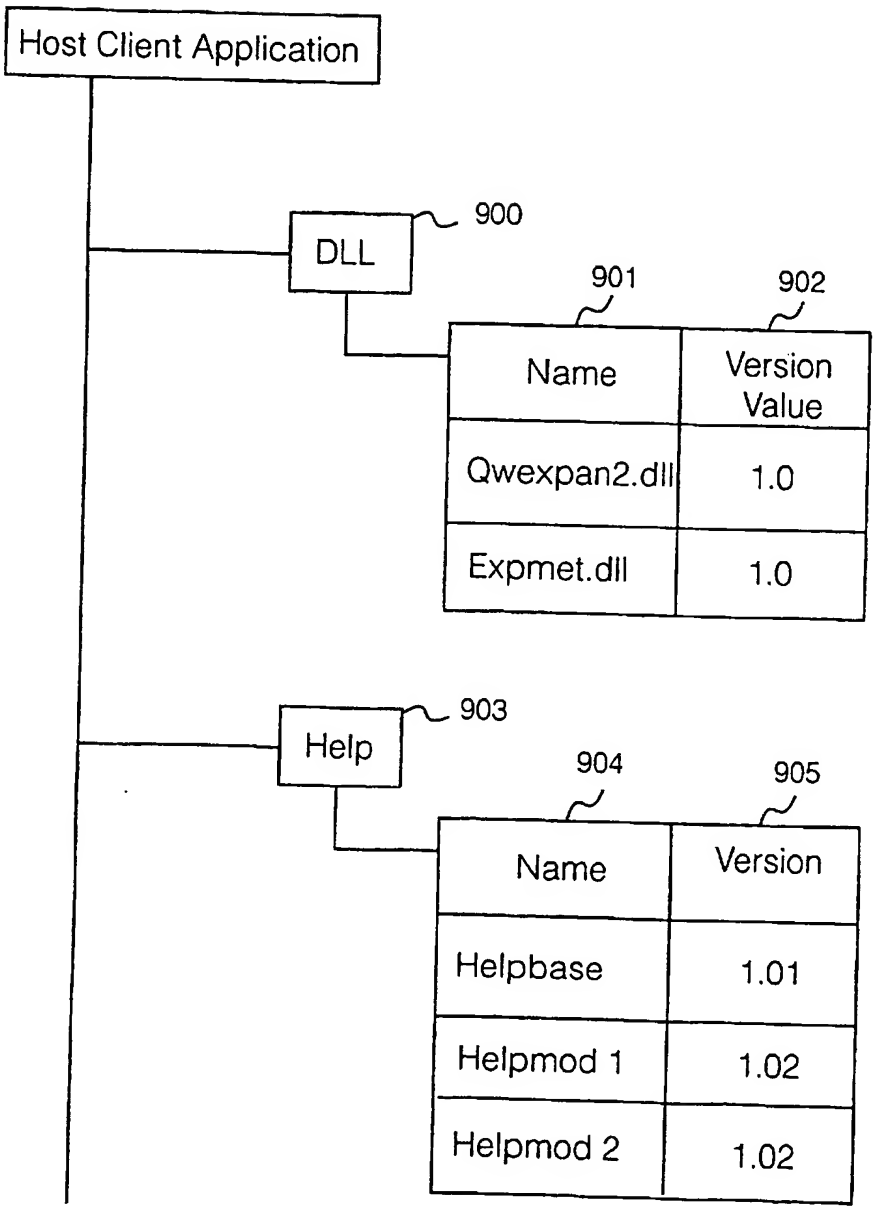
Host Client Application
DLL
900
901
902
Name
Version Value
Qwexpan2.dll
1.0
Expmet.dll
1.0
Help
903
904
905
Name
Version
Helpbase
1.01
Helpmod 1
1.02
Helpmod 2
1.02

Fig. 9

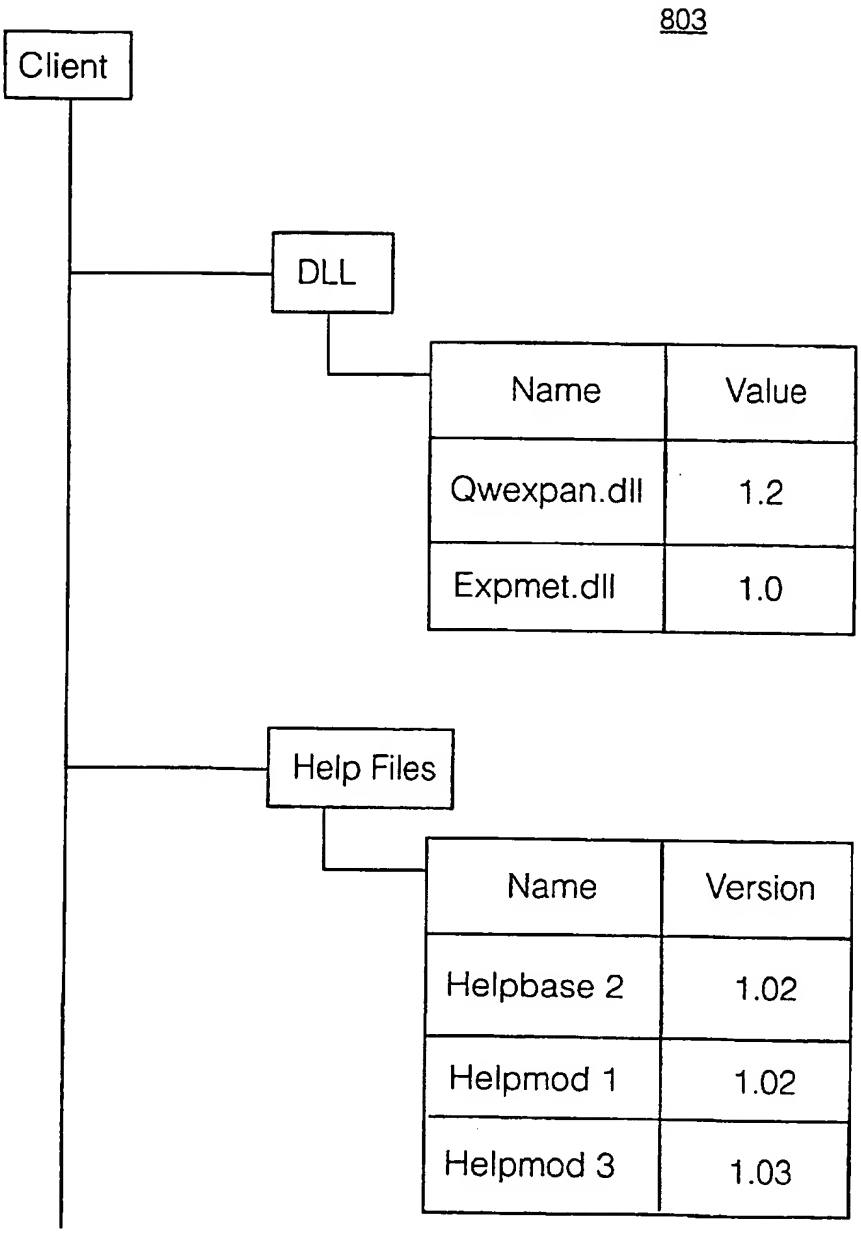
803
Client
DLL
Name
Value
Qwexpan.dll
1.2
Expmet.dll
1.0
Help Files
Name
Version
Helpbase 2
1.02
Helpmod 1
1.02
Helpmod 3
1.03

Fig. 10

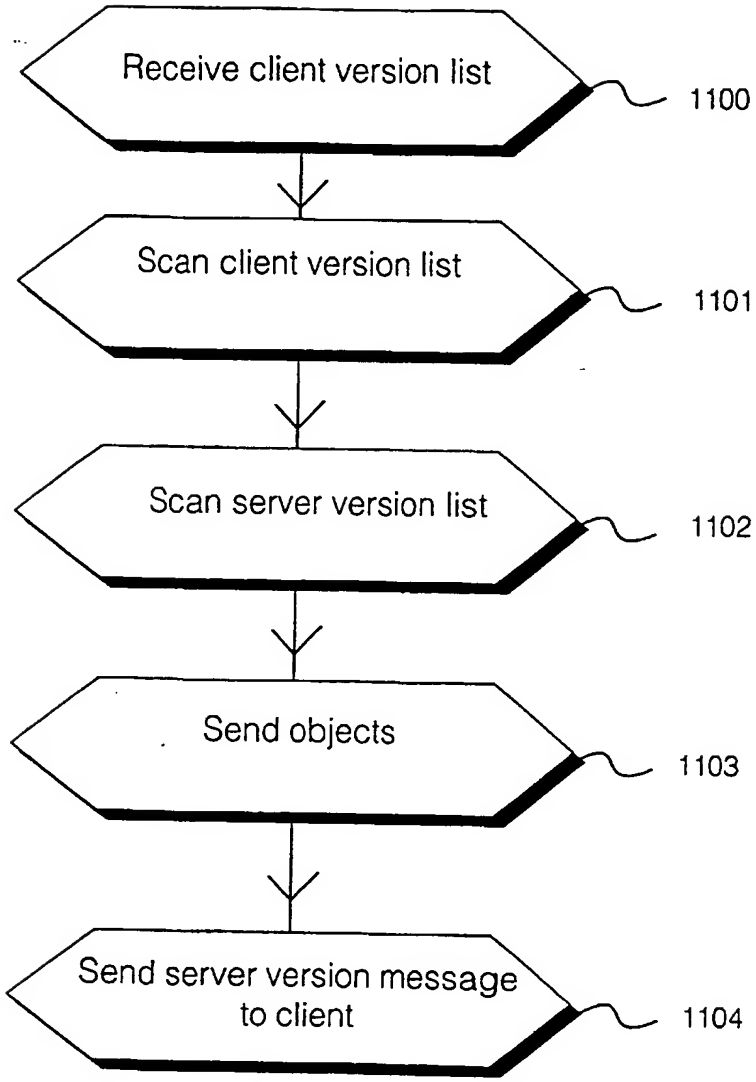
Receive client version list
1100
Scan client version list
1101
Scan server version list
1102
Send objects
1103
Send server version message to client
1104

Fig. 11

1100

1101 Type	1102 Name	1103 Instruction
DLL	Qwexpan.dll	Remove
DLL	Qwexpan2.dll	Add
DLL	Expmet.dll	Retain
HLP	Helpbase.hlp	Remove
HLP	Helpbase2.hlp	Add
HLP	Helpmod1.hlp	Retain
HLP	Helpmod2.hlp	Remove
HLP	Helpmod3.hlp	Add

Fig. 12

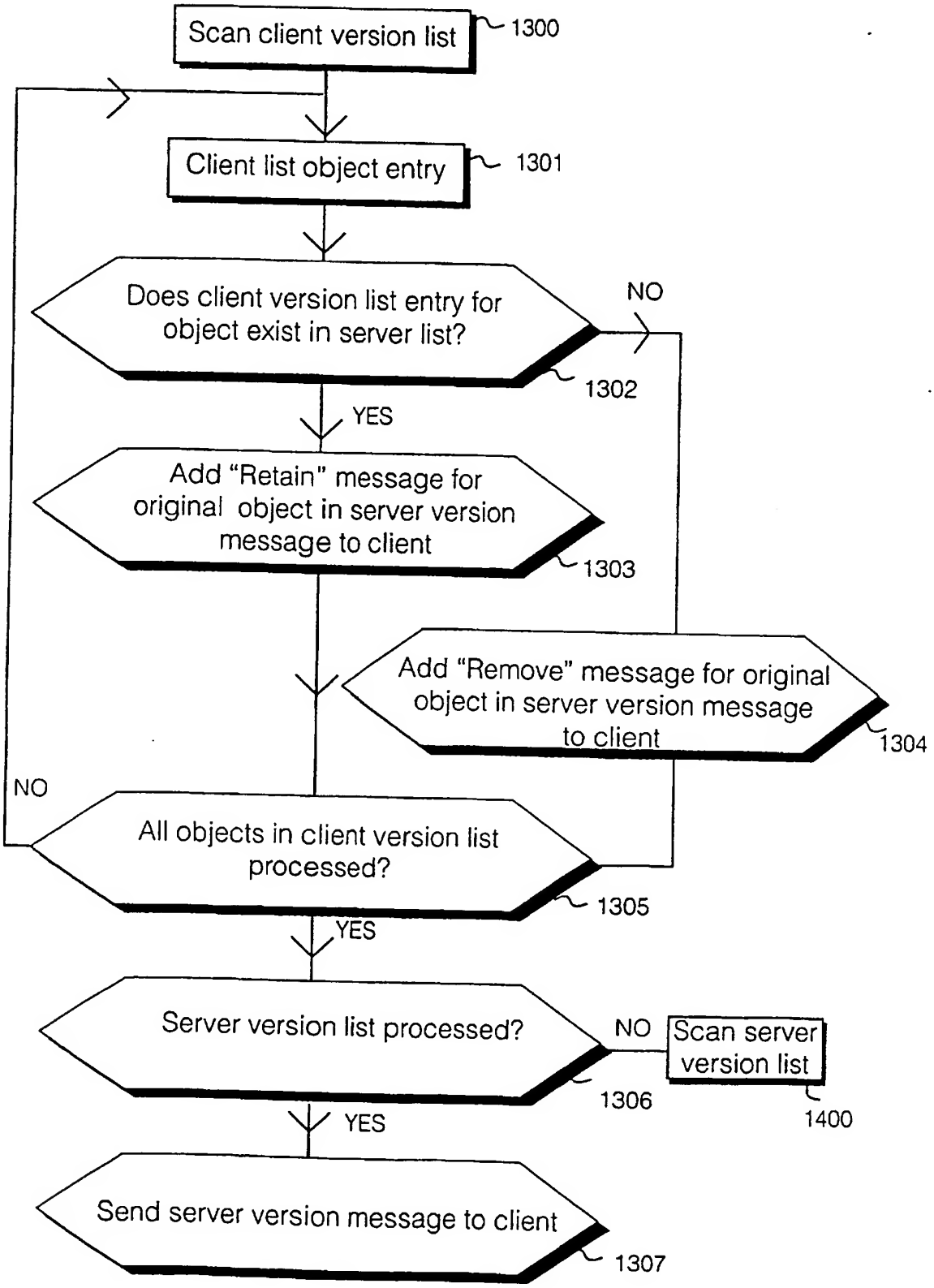
Scan client version list
1300
Client list object entry
1301
Does client version list entry for object exist in server list?
NO
1302
YES
Add "Retain" message for original object in server version message to client
1303
Add "Remove" message for original object in server version message to client
1304
NO
All objects in client version list processed?
1305
YES
Server version list processed?
NO
Scan server version list
1306
1400
YES
Send server version message to client
1307

Fig. 13

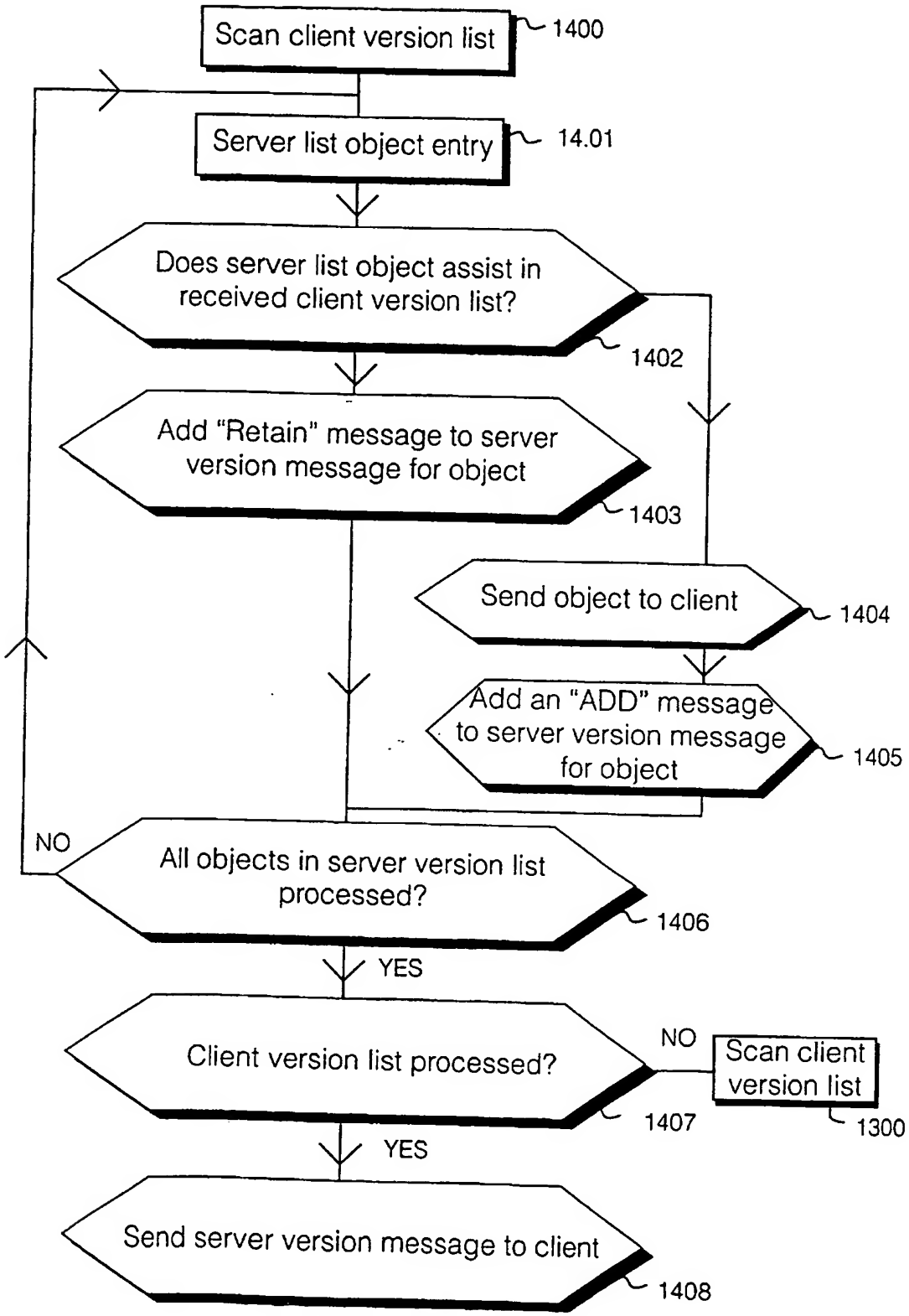
Scan client version list
1400
Server list object entry
14.01
Does server list object assist in received client version list?
1402
Add "Retain" message to server version message for object
1403
Send object to client
1404
Add an "ADD" message to server version message for object
1405
NO
All objects in server version list processed?
1406
YES
Client version list processed?
NO
Scan client version list
1300
1407
YES
Send server version message to client
1408

Fig. 14

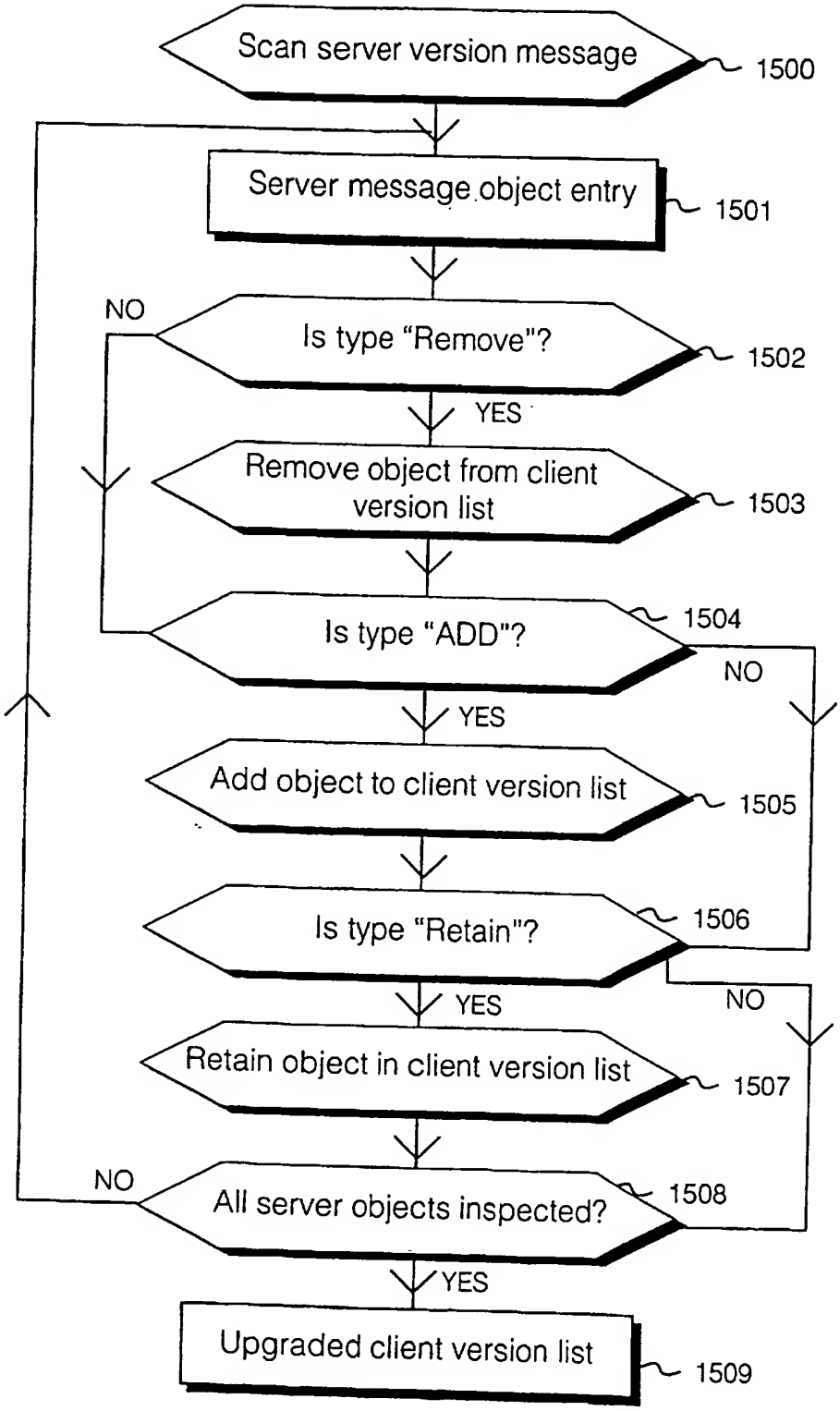
Scan server version message
1500
Server message object entry
1501
NO
Is type "Remove"?
1502
YES
Remove object from client version list
1503
Is type "ADD"?
1504
NO
YES
Add object to client version list
1505
Is type "Retain"?
1506
YES
NO
Retain object in client version list
1507
NO
All server objects inspected?
1508
YES
Upgraded client version list
1509

Fig. 15

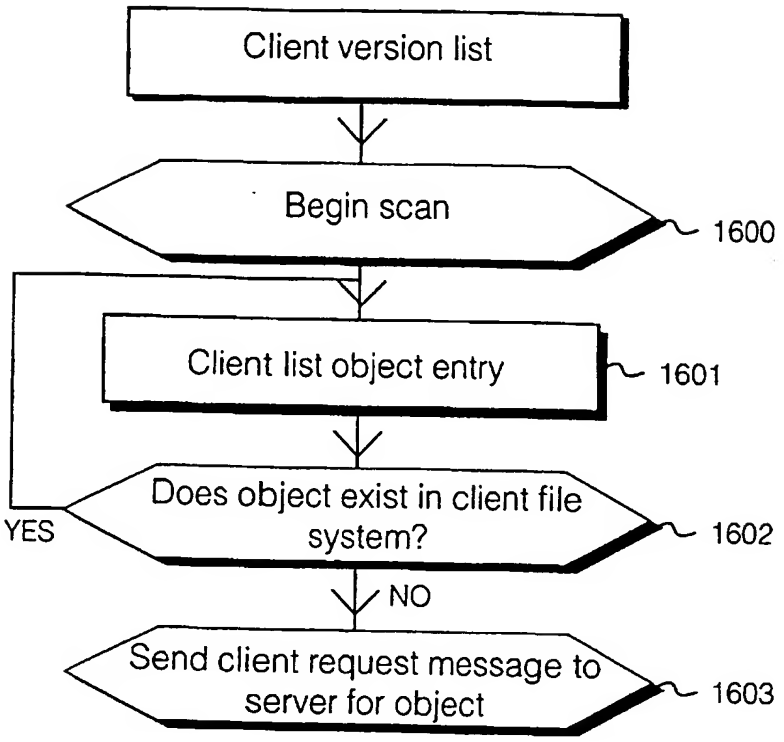
Client version list
Begin scan
1600
Client list object entry
1601
Does object exist in client file system?
1602
YES
NO
Send client request message to server for object
1603

Fig. 16